Pravila kodiranja

U okviru ove lekcije govorićemo o pravilima kodiranja, odgovorima na to zašto su pravila kodiranja važna u organizaciji posla, kao i o prednostima koje nam organizovano i standardizovano pisanje koda donosi. Na samom kraju, fokusiraćemo se na pravila kodiranja u Pythonu prema PEP 8 vodiču za stilizovanje Python koda.

Pravila kodiranja (coding conventions) predstavljaju skup saveta i pravila za pisanje koda u određenom programskom jeziku. U okviru ovih pravila pronalazimo stilove programiranja, primere dobre prakse, kao i odgovarajuće metode za svaki aspekt pisanja u okviru programskog jezika. U okviru ovih pravila uobičajeno je da pronađemo pravila za organizaciju fajlova, indentaciju, komentare, deklaracije, definicije pravilnog korišćenja praznih prostora i, generalno, primere dobre prakse.

Svim vrstama programera se preporučuje da prate pravila kodiranja za svoj programski jezik, što će im pomoći da postignu bolju čitiljvost koda, a ujedno i da olakšaju kasnije održavanje programa. Pravila kodiranja su važna samo u slučaju kada ljudi obavljaju održavanje i pregled softvera. U slučaju da se radi automatizovano testiranje, proces sa kojim ćemo se susresti kasnije u ovom kursu, ta pravila nemaju uticaj na čitljivost jer kod tada čita računar.

Pravila kodiranja služe kao vodič za postizanje strukturnog kvaliteta koda. Veoma često, programe razvijaju timovi inženjera i programera i programski kod često menjaju različiti ljudi. U većini slučajeva, kod se samo jednom piše, ali više puta čita, bilo da to čine kolege u timu, menadžeri projekta, pa čak i drugi razvojni timovi u kasnijoj primeni programa. Stoga je vrlo važno da za kratko vreme možemo da razumemo šta konkretan kod radi – dakle, da mu povećamo čitljivost.

Ciljevi i prednosti uvođenja standarda u pisanje izvornog koda su:

- povećanje čitljivosti i postizanje identičnog pristupa pisanju koda kod celokupnog razvojnog tima;
- sprečavanje i lakše uočavanje semantičkih grešaka u programu – ukoliko je kod bespotrebno kompleksan, postoje veće šanse da bude ranjiv na greške; pravila kodiranja pomažu u procesu razvoja softvera koji je manje kompleksan i samim tim smanjuju šanse pojave grešaka iz ovih razloga;
- olakšavanje debagovanja i održavanja – ukoliko se prate pravila kodiranja, kod je, kako smo naveli, uvek identičan po strukturi, što prvenstveno olakšava pronalaženje i ispravljanje bugova; ujedno, održavanje je olakšano jer svako može menjati kod nezavisno od vremena koje je prošlo ili od člana razvojnog tima;
- povećanje produktivnosti programera koji rade u timovima – često navodimo da razvojni timovi mogu provesti mnogo vremena u ispravljanju grešaka koje su mogle biti sprečene; implementacija pravila kodiranja ujedno predstavlja i mehanizam za sprečavanje ovih grešaka, što će direktno uticati na produktivnost programera, jer će oni sada biti fokusirani na zadatke koji vode kompletiranju projekata, umesto na ispravljanje grešaka;
- smanjenje troškova razvoja – programski kod kreiran uz primenu pravila kodiranja je čist i pregledan, što otvara mogućnost korišćenja delova koda i u nekim narednim projektima, što drastično smanjuje troškove razvoja.

Pravila kodiranja mogu biti definisana u okviru jednog dokumenta kojim se vodi ceo tim, ali i npr. cela kompanija. Primer jednog takvog standarda možete videti na sledećem linku:

https://cse.buffalo.edu/~rapaport/code.documentation.Excerpts.pdf

Ovaj standard je razvijen za potrebe razvojnih timova Univerziteta Buffalo i projekata u kojima se koristi programski jezik C. Na 30 strana dokumenta je definisan svaki aspekt koda sa preporukama i primerima.

Alternativa dokumentaciji na nivou organizacija mogu biti neformalni dogovori između članova tima, pa čak i na nivou pojedinca, gde mi sami definišemo pravila koja nisu definisana u dokumentu, u samom kodu kroz pristup koji primenjujemo. Ono što je važno zapamtiti jeste da nijedan kompajler/interpreter neće zahtevati od vas da poštujete neko pravilo kodiranja – za njih je važno samo da poštujete sintaksu jezika.

Pre nego što nastavimo, važno je da znate da, iako postoje neka univerzalna pravila za svaki od programskih jezika, zavisno od tipa projekta koji radite i kompanije za koju budete radili, uvek će postojati neke varijacije u samim pravilima. U nastavku ćemo prikazati osnovna pravila koja se primenjuju za svaki programski jezik i standard PEP 8, koji je zvanično preporučen za programski jezik Python.

Pogledajmo sada samo neka od opštih pravila o kojima uvek treba voditi računa prilikom pisanja koda:

- Nastojati da se kod deli u sekcije, gde bi blok koda predstavljao jedan izdvojen paragraf.
- Indentacija (uvlačenje teksta, tj. koda) treba da postane stvar navike. Naročito je obavezna na početku i kraju struktura, kao i u unutrašnjosti struktura gde je namenjen prostor za blok koda.
- Na samom početku rada odrediti jedan od načina imenovanja promenljivih i voditi se njime tokom celog trajanja projekta. Takođe, za svaki novi podatak poželjno je komentarom opisati koja će biti njegova svrha.
- Funkcije i metode uvek nazivati prema onome što će obavljati.
- Kod bi trebalo da bude dovoljno jasno napisan da ga čak i nakon dužeg vremenskog perioda možete sa lakoćom čitati i ponovo razumeti kako funkcioniše.
- Prilikom rada uvesti jedan sistem komentarisanja koda i praktikovati ga tokom celog projekta.
- Funkcije i strukture koje su previše nepregledno napisane i teške za razumevanje usled loše čitljivosti je poželjno izbegavati.

Ovo su neka osnovna pravila. U nastavku, fokusiraćemo se na njihovo pojašnjenje i detaljnije obrazloženje sa primerima upotrebe programskog jezika Python.

Uvod u PEP 8

PEP – skraćeno od Python Enhancement Proposal – predstavlja skup pravila koje možete koristiti u svrhu bolje organizacije i čitljivosti Python koda. U okviru ove lekcije fokusiraćemo se na najvažnije delove i pravila pisanja koda, a celokupan dokument koji sadrži sve preporuke za pisanje Python koda, *PEP 8 Style Guide for Python Code*, možete pronaći na sledećem linku: https://www.python.org/dev/peps/pep-0008/

Konvencija imenovanja (naming convention)

Konvencijom imenovanja definišemo način na koji ćemo pisati nazive pre svega promenljivih, funkcija i klasa. Najpopularnije konvencije su:

- **Camel Case:** U okviru Camel Case konvencije svako prvo slovo reči je napisano velikim slovom, ne smeju se koristiti razmaci i simboli u nazivima. Primeri upotrebe ove konvencije mogu biti: UserAccount, MyClass, FirstNumber. U okviru zajednice programera postoji i česta varijacija ove konvencije, gde naziv započinje malim slovom, a svaka naredna reč ima veliko prvo slovo. Npr: fileName, userAccount, myClass. Ovu varijaciju kompanija Microsoft koristi kada govori o Camel Case konvenciji.
- **Pascal Case:** Ova konvencija je nastala sa korišćenjem Pascal programskog jezika; smatra se potkonvencijom Camel Case. U ovoj konvenciji svaka reč počinje velikom slovom, dakle npr. UserAccount, ali u ovoj konvenciji nije praksa započeti prvu reč sa malim slovom, dakle nije dozvoljeno userAccount.
- **Snake Case:** U Snake Case konvenciji reči se mogu razdvajati, ali za razdvajanje se koristi karakter donje crte. Primera radi, nazivi mogu biti: user_account, error_message, account_balance.
- **Kebab Case:** Slična Snake Case konvenciji, ali umesto karaktera donje crte koristimo crticu (hyphen). Dakle, nazivi bi izgledali ovako: user-account, error-message, account-balance.
- **Screaming Case:** Ova konvencija se odnosi na pisanje naziva velikim slovima. Primer: USERACCOUNT, ERROR_MESSAGE.

Python, naravno, podržava svaku od ovih konvencija. Pored strogog pridržavanja jedne od ovih konvencija, možete koristiti i konvenciju koja je definisana i u samom PEP 8 vodiču. Ovu konvenciju možete videti na tabeli u nastavku.

Tip	Konvencija nazivanja	Primer
Funkcije	Koristiti mala slova za reč/reči, koristiti donje crte radi unapređenja čitljivosti koda	function my_function
Promenljive	Koristiti mala slova za reč/reči, koristiti donje crte radi unapređenja čitljivosti koda	x, var, my_variable
Klase	Primenjivati Camel Case konvenciju	Model, MyClass
Metode	Koristiti mala slova za reč/reči, koristiti donje crte radi unapređenja čitljivosti koda	class_method, method
Konstante	Poželjno korišćenje velikih slova za formiranje reči, koristiti donje crte radi unapređenja čitljivosti koda	CONSTANT MY_CONSTANT MY_LONG_CONSTANT
Moduli	Koristiti kratke reči, pisati malim slovima, razdvajanje reči karakterom donje crte	module.py, my_module.py
Paketi	Koristiti kratke reči, pisati malim slovima i ne razdvajati reči	Package mypackage

Tabela 4.1. Konvencija nazivanja prema PEP 8

Ovo su neke od čestih konvencija nazivanja i primeri njihovog korišćenja. Ali, ovo nije jedina stavka pisanja čitljivih naziva; vrlo je važno koja slova i reči koristimo.

Dodeljivanje naziva

Dodeljivanje naziva je možda najlošije standardizovana oblast programskih jezika. U nastavku, pružićemo specifikaciju za Python, ali naravno, neke od ovih napomena važe i za većinu drugih programskih jezika. Jedan od ključnih faktora za razumevanje koda je upotreba smislenih naziva promenljivih, konstanti, struktura i sl. Upotrebom razumljivih imena nastaje *samodokumentujući* kod, tj. kod koji je razumljiv sam po sebi bez potrebe za pisanjem dodatnih komentara.

Pogledajmo sledeći primer:

```
# Bad Example
a = 'John Snow'
b, c = a.split()
print(c, b, sep=', ')
# Result is Snow, John
```

Sigurno ste do sada pokušali da koristite što kraće nazive za promenljive, najčešće da što brže napišete neki deo koda. U ovom kratkom primeru, promenljivu a koristimo za čuvanje stringa; u promenljive b i c upisujemo po jednu reč nakon upotrebe split() funkcije nad stringom. Kada prvi put napišemo ovaj program, sve bi delovalo u redu, sve će raditi pravilno. Ali da se ovome programu vratimo nakon određenog vremenskog perioda, morali bismo da ispratimo svaku liniju i utvrdimo gde je tačno koji od podataka premešten i koja je svrha te promenljive. Ovaj problem bi bio još veći da je ovaj mali blok koda deo programa od par hiljada linija koda. Ovo je upravo primer poštovanja svih sintaksnih pravila jezika, ali ujedno i pisanja koda koji nije dovoljno čitljiv.

Složićete se da isti primer mnogo bolje izgleda ovako:

Radno okruženje

```
# Recommended
name = 'John Snow'
first_name, last_name = name.split()
print(last_name, first_name, sep=', ')
# Result is Snow, John
```

Objašnjenje:

Bolja praksa za imenovanje promenljivih jeste definisanjem imena koja asociraju na vrednosti koje trebaju da sadrže kao vrednosti. Npr. first_name će sadržati ime a last_name prezime koje se dobije od promenljive name.

Pored izrazito kratkih naziva, generalno treba izbegavati imena duža od 15 karaktera, jer ona smanjuju čitljivost. Iako možda deluje da na ovaj način postižemo bolje dokumentovanje samog koda, to ipak nije slučaj. Npr. ako imamo promenljivu:

```
SetButtonLabelText = 'Start'
```

ona izdvojeno deluje odlično, jasno je da će tekst biti iskorišćen za naziv dugmeta, ali problemi nastaju kada počnemo da je koristimo u samom programu, pa npr. želimo da iskoristimo tri promenljive kao parametre funkcije koja će generisati jedno dugme u interfejsu:

def generate_button(SetButtonLabelText,setButtonPrimaryColor, setButtonShadowColor):

Iako je vrlo jasna svrha svake promenljive, ova linija zauzima 84 karaktera od poželjnih 79 karaktera prema PEP 8 – dakle, ovo se već sada smatra loše čitljivom linijom.

Pored dužine naziva, izbegavati i imena koja su veoma slična ili se razlikuju samo u jednom karakteru. Vrlo često možemo videti nešto poput:

```
tax1 = 112
tax2 = 23

print(tax1-tax2)
```

Dakle, u ovome kodu vidimo da se obračunavaju neki porezi. Samo autor koda zna šta je tax1, a šta je tax2 i vrlo je moguće da bez dokumentacije ni sam autor nakon nekog vremena neće znati šta je bio tax1, a šta tax2.

Stoga, sigurno bi bilo bolje da smo nazive dodelili na sledeći način:

Radno okruženje

```
incomeTax = 112
trafficTax = 23

print(incomeTax-trafficTax)
```

Napišite program koji za unetu vrednost broja kilometara kao celog broja ispisuje zadatu vrednost u miljama. Jedan kilometar je 0.621371 milja.

Pitanje

Koji standard je zvanično preporučen za programski jezik Python?

- **PEP 8**
- PEF 8
- PEP 9

Objašnjenje:

PEP 8 je zvanično preporučeni standard za programski jezik Python.

Komentarisanje koda

Kao što smo i ranije navodili, komentarisanje koda je dobra praksa. Komentarisanje omogućava lakše razumevanje koda kako vama tako i drugim članovima tima. Sa korišćenjem IDE i drugih alata, komentarisanje koda možemo iskoristiti na više načina. Prvenstveno, preporučljivo je prilikom pisanja neke nove funkcije ili metode u komentaru ukratko opisati šta će ona raditi. Pored toga, možete navesti koje ulazne parametre koristi, koju će vrednost vratiti na mesto poziva i slično. Pored navedenog, uloga svakog paketa ili klase bi trebalo da bude sumirana u komentarima fajlova.

Na primer, u slučaju funkcije koja sabira dve vrednosti, komentar bi mogao da izgleda ovako:

```
# this function will be used for the addition of two variables
def add(firstNumber, secondNumber):
    return firstNumber + secondNumber
```

Komentare možemo pisati na samom početku dokumenta (u zaglavlju) sa namenom da identifikujemo autora i namenu programa i pružimo informacije o autorskim pravima, verziji programa, poslednjim izmenama i sl. Template komentara zaglavlja može izgledati ovako:

```
# ************************************************************
#
# Name: Python Naming Convention Example
# Description: This Program does nothing
# Author: John Snow
# Created on: 20. 01 . 2022
# Last change on: 31. 05 2022.
#
# Important: Generic content
#
# ************************************************************
```

Odličan realan primer korišćenja komentara, kao i njihove upotrebe u vidu dokumentovanja, možete videti na primeru Python Developer Guidea, koji je dostupan na GitHub platformi, na linku: https://github.com/python/devguide/blob/master/conf.py

Važno je ne preterati sa komentarima u kodu. Ne treba komentarisati očigledne stvari, jer će previše komentara opet učiniti kod manje čitljivim. Recimo, sledeći kod ne iziskuje upotrebu komentara za dodatno pojašnjenje:

```
for x in range(6): # for loop starts here
    print(x)     # showing the result
```

Još jedan primer dobre prakse je grupisanje koda u blokove, gde svaki blok koristi komentar za dodatno pojašnjenje. Na primer:

```
# this function will be used for the addition
def Add():
    # logic here

#  this function will be used for deletion
```

```
def Delete():
    # logic here
```

Napravite listu L sa 3 proizvoljna cela broja. Definišite promenljivu Add u kojoj se dodaje prosleđeni broj na kraj liste a kao povratnu vrednost postaviti listu. Funkcija Delete bi trebala da briše poslednji element liste, takođe je potrebno da vratite listu kao povratnu informaciju. Pozvati funkcije Add i Delete nad zadatom listom. Pišite komentare za svaki deo koda.

Uvlačenje linija koda (indentacija)

Uvlačenje linija koda je potrebno za klase, funkcije, metode, petlje, uslove i liste. Može se koristiti tabulator ili prazna mesta, ali koja god opcija da se koristi – programer je se stalno mora pridržavati. Za Python 2.x verziju nije bilo bitno koji se metod koristi, ali je od Python 3.x verzije poželjan metod korišćenje praznih mesta, jer tabulator ne predstavlja isti broj karaktera na svakom sistemu (negde je to četiri mesta, negde dva, negde više). Zbog toga se za Python 3.x preporučuju četiri mesta po nivou uvlačenja. Ako se piše dugačak izraz, najbolje je držati ga vertikalno poravnatim.

Za kraj, kad je reč o uvlačenju linija koda, ali generalno i o navikama pisanja, trebalo bi izbegavati duboko ugnežđivanje strukture. U nastavku možete videti primer strukture koju ne bi trebalo upotrebljavati:

```
def login(email, password, new_password, confirm_new_password):
    if(email and password and new_password and confirm_new_password):
        if(new_password == confirm_new_password):
            if(login(email, password)):
                if(set_password(email, new_password)):
                    return TRUE
```

Napišite program u koji se unosi preko input funkcije jedan string. Ispitati da li je prvi karakter ‘a‘, ako jeste ispitati da li je drugi karakter ‘b‘, ako jeste ispitati da li je treći karakter ‘c‘. Ako je tačno ispisati Tačno a u svakom ostalom slučaju ispisati Netačno.

Radno okruženje

Validacija stila pisanja

Smernice su odličan način za standardizaciju pisanja koda i programer bi trebalo da ih se pridržava koliko god je moguće. Postoje i rešenja koja automatski skeniraju i traže greške u stilu pisanja. Za Python 3 postoje zvanični, ali i eksterni moduli koji proveravaju stil pisanja. Jedan od primera eksternih modula je python3-pycodestyle, koji se može instalirati pomoću komande pip3 install python3-pycodestyle. Lakši način za proveru stila pisanja je korišćenje besplatnog onlajn alata po imenu PEP8 online check. Nakon unošenja našeg koda, možemo dobiti povratnu informaciju o njegovoj ispravnosti, nalik izveštaju na slici 4.1.

PEP8 online

Check your code for PEP8 requirements

Check results

Save ▾ Share

Code	Line	Column	Text
E401	12	12	multiple imports on one line
E231	14	23	missing whitespace after ','
E231	14	30	missing whitespace after ','
W291	14	72	trailing whitespace
E128	15	5	continuation line under-indented for visual indent
E231	15	42	missing whitespace after ','
E231	15	55	missing whitespace after ','
E501	15	80	line too long (111 > 79 characters)
E231	15	96	missing whitespace after ','
E251	21	31	unexpected spaces around keyword / parameter equals
E251	21	33	unexpected spaces around keyword / parameter equals
E101	23	1	indentation contains mixed spaces and tabs

Slika 4.1. Rezultati analize stila koda

Ovu lekciju smo posvetili još jednom podsećanju na važnost pisanja preglednog koda; upoznali smo pravila fokusirana na čitljivost koda. U narednoj lekciji počinjemo sa radom u oblasti testiranja i krećemo sa testiranjem najmanjih celina programa – testiranjem jedinica.

Rezime

- Pravila kodiranja (coding conventions) predstavljaju skup saveta i pravila za pisanje koda u određenom programskom jeziku. U okviru ovih konvencija uobičajeno je da pronađemo pravila za organizaciju fajlova, indentaciju, komentare, deklaracije, definicije pravilnog korišćenja praznih prostora i, generalno, primere dobre prakse.
- Uvođenje standarda u kodiranju donosi veliki broj prednosti; pre svega, dovodi do povećanja čitljivosti i postizanja identičnog pristupa pisanju koda kod celokupnog

razvojnog tima. Greške se mogu sprečiti i lakše uočiti, a ujedno je olakšan proces debagovanja i održavanja.

- Konvencijom imenovanja definišemo način na koji ćemo pisati nazive pre svega promenljivih, funkcija i klasa. Najpopularnije konvencije su: Camel Case, Pascal Case i Snake Case.
- Komentarisanje koda omogućava lakše razumevanje koda, kako vama tako i drugim članovima tima. Sa korišćenjem razvojnih okruženja i drugih alata, komentarisanje koda možemo iskoristiti na više načina.
- Smernice u kodiranju su odličan način za standardizaciju pisanja koda i programer bi trebalo da ih se pridržava koliko god je to moguće. Postoje i rešenja koja automatski skeniraju i traže greške u stilu pisanja. Za Python 3 postoje zvanični, ali i eksterni moduli koji proveravaju stil pisanja.